Stocks & Crypto SQL Show #3

Build fast dashboards with TimescaleDB (+ reduce storage costs)

Attila Toth, Developer Advocate at Timescale

Timescale

Agenda

01 Continuous aggregates to fuel dashboards

02 Create the dashboard (Apache Superset)

03 Use compression to reduce storage costs

04 Next steps

01

Continuous aggregates to fuel dashboards

Continuous aggregates to fuel dashboards

Continuous aggregates is materialized view for time-series data with continuous and incremental refresh and faster aggregate queries.

Why use continuous aggregates for dashboards?

- Always see up-to-date information (even if the data is not yet materialized)
- Fast queries → fast-loading charts
- Easy and fast filtering in your dashboard tool
- While the view is getting refreshed, your dashboard will keep working as usual
- Data gets refreshed faster

Continuous aggregates → good query performance → fast dashboards

Let’s create a real continuous aggregate!

Continuous aggregates definition

Create a continuous aggregate that will fuel our dashboard:

```
CREATE MATERIALIZED VIEW demo_collections_daily
WITH (timescaledb.continuous) AS
SELECT
collection_id,
time_bucket('1 day', time) AS bucket,
COUNT(*) AS volume,
SUM(total_price) AS volume_eth,
MAX(total_price) AS max_price,
MIN(total_price) AS min_price,
AVG(total_price) AS avg_price
FROM table
WHERE payment_symbol = 'ETH'
GROUP BY bucket, collection_id;
```

02

Create the dashboard (Apache Superset)

03

Use compression to reduce storage costs

Use compression to reduce storage costs

- Compression is a cornerstone feature of TimescaleDB
- Converts multiple rows into one row (based on one or multiple *segmentby* columns)
- Stores multiple rows' data inside one array
- Less rows = less disk space required → reducing disk storage costs

Enable compression

Enable compression:

```
ALTER TABLE temperature SET (
  timescaledb.compress,
  timescaledb.compress_segmentby = 'device_id'
);
```

With *orderby* columns:

```
ALTER TABLE example SET (
  timescaledb.compress,
  timescaledb.compress_segmentby = 'device_id',
  timescaledb.compress_orderby = 'time DESC'
);
```

Manual compression and automation policy

Manually compress chunks that are older than 1 week, but newer than 3 weeks:

```
SELECT compress_chunk(i, if_not_compressed=TRUE)
FROM show_chunks('temperature',
                 older_than = now() - interval '1 week',
                 newer_than = now() - interval '3 weeks') i;
```

Set up a compression policy to compress chunks older than 60 days:

```
SELECT add_compression_policy('temperature', INTERVAL '60 days');
```

Thank you for watching!
Next steps:

01

Learn more about TimescaleDB:

docs.timescale.com

02

Join the TimescaleDB community:

slack.timescale.com

03

Tweet us:

@TimescaleDB

@AttilaTothDev

We are hiring! → timescale.com/careers